Making Abstraction Concrete:
An Architect's Catalogue of Abstraction Patterns

Bran (The Terminator) Selic

Malina Software Corp., Canada
Simula Research Labs, Norway
Zeligsoft (2009) Ltd., Canada
University of Toronto, Canada
Carleton University, Canada

selic@acm.org

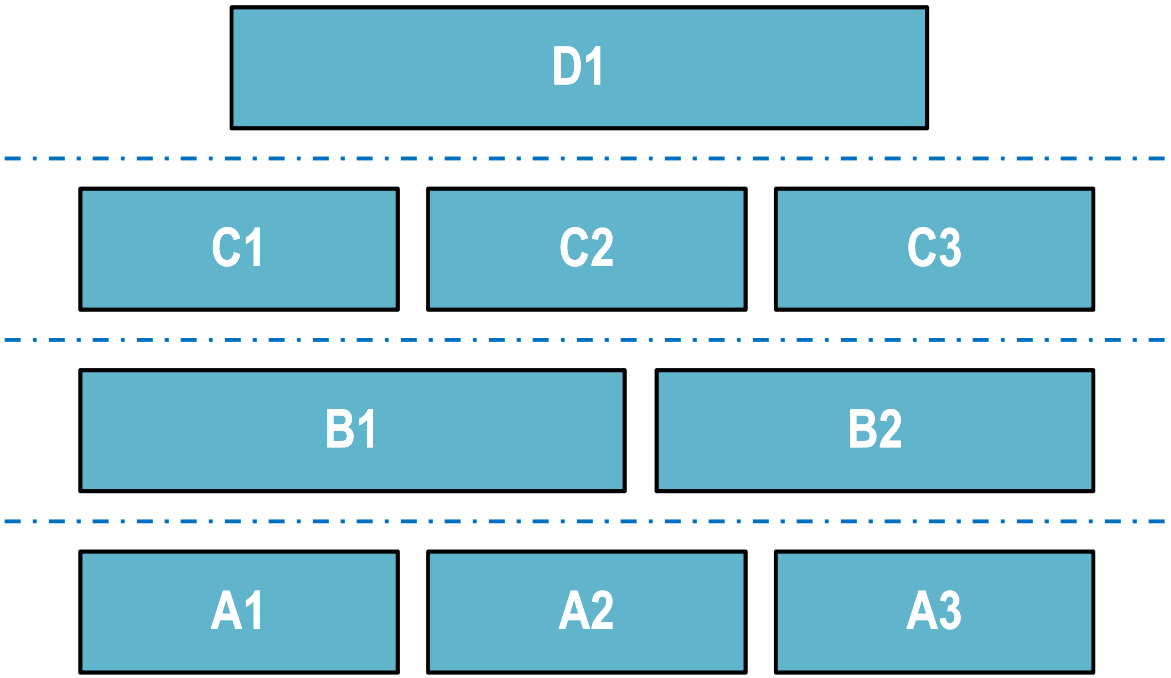
From a Keynote by J. Naughton

- Advice given by an Emeritus Professor regarding keynotes:

> *"Don't do it! Giving a keynote means you are washed-up has-been."*

A "Wake Up" Question

- Why is the following often-used form usually a misleading and inaccurate representation of the architecture of a "layered" software system?



Answer coming up soon!

Overview

- Part I: On abstraction and modeling
- Part II: A starter catalogue of abstraction patterns for system architects

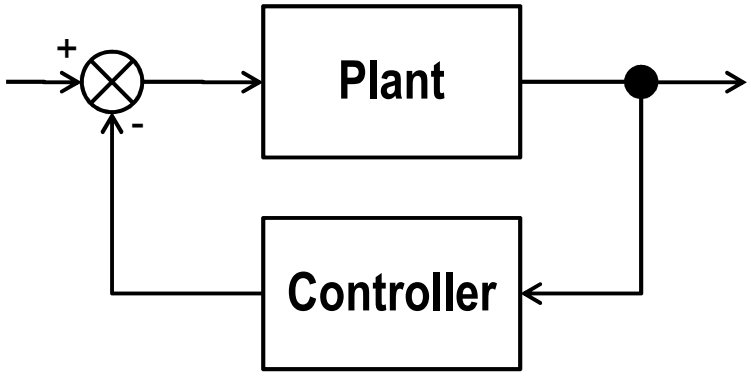
What is "Abstraction"?

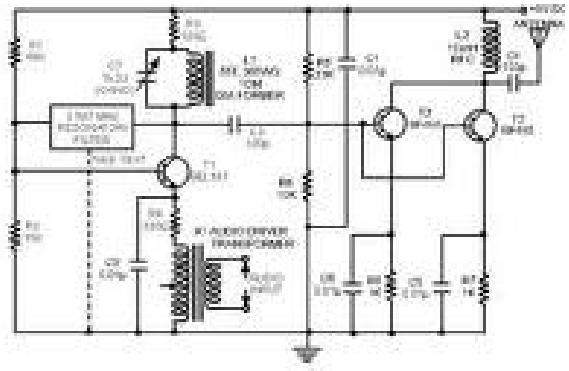
- An *intellectual defence mechanism* for coping with overwhelming complexity
  - Often, our only mechanism

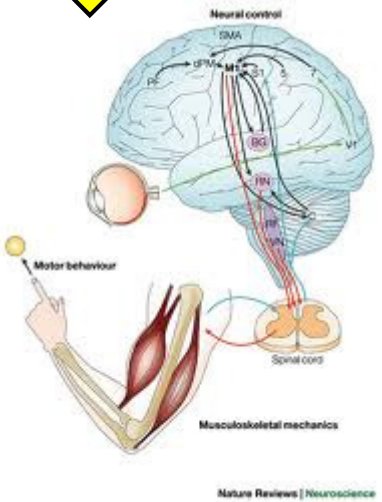
WARNING: Don't confuse abstraction with reality: It doesn't always work

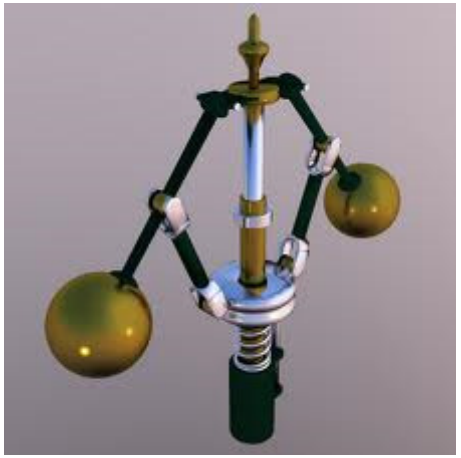
An Eminently Successful Examplar

- The model of feedback control:









Abstraction: Definitions I Don't Like

- ABSTRACTION: *A process by which "higher" concepts are derived from the usage and classification of literal ("real" or "concrete") concepts, first principles, and/or other abstractions* [Wikipedia]
- ABSTRACTION (computer science): *a mechanism and practice to reduce and factor out details so that one can focus on a few concepts* [Wikipedia]

Technical Approaches to Abstraction

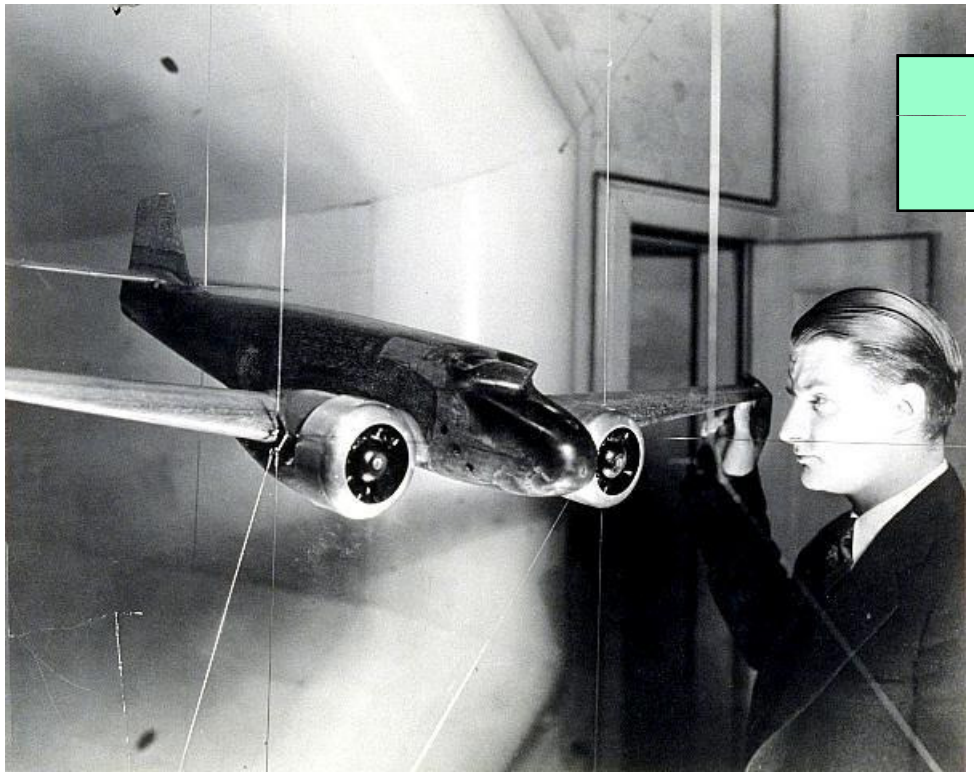
- **Relaxionist: By loosening constraints**
  - Broadens scope of coverage
  - E.g., going from "Square" to "Geometrical Shape"
- **Reductionist: By *removing or merging* "irrelevant" detail**
  - Relevance is a function of viewpoint (set of concerns)

Abstraction: A Crucial Skill for Architects

- **It seems to be less common than one might expect**
  - ...particularly among software people
  - The ability to see beyond the technology
- **There are many programmers but few "architects"**
  - Agile development and architecture perceived as mutually exclusive by many practitioners
  - E.g., M. Fowler: "Is design dead?"
- **Can abstraction be taught?**
  - …I think so, but some people are naturally better at it

Abstraction: My Favoured Definition

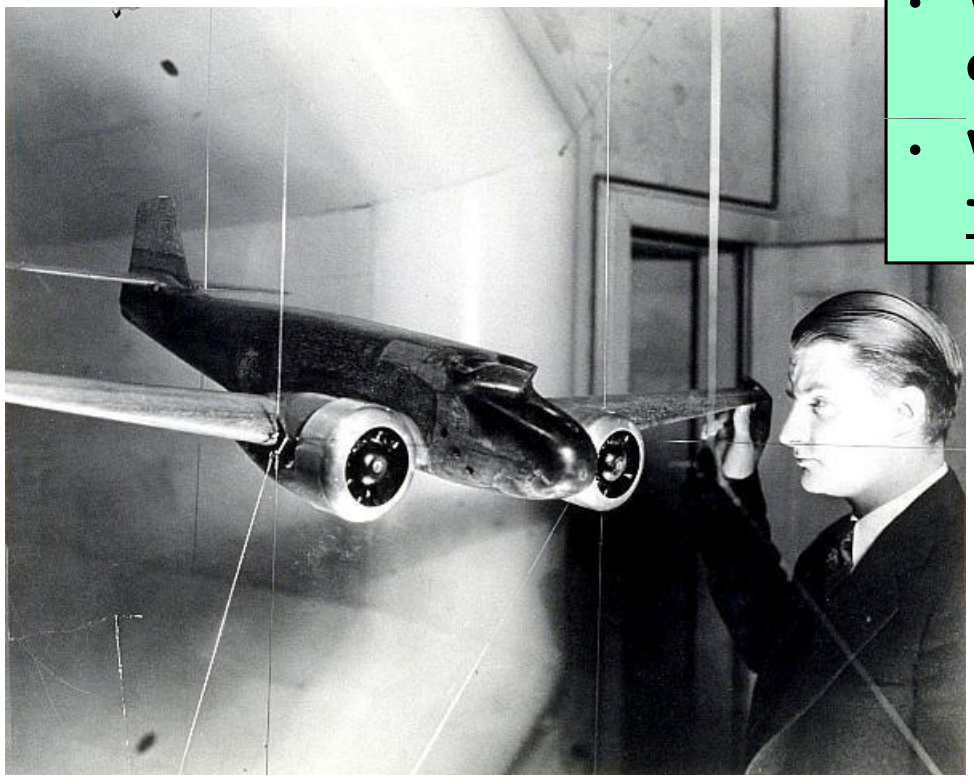
- **ABSTRACTION :** ***Selective reduction of information of a system which preserves its salient properties relative to a given set of concerns***
  - Refinement is the inverse process



In engineering:
abstraction ⇒ modeling

Engineering Models

- ENGINEERING MODEL: *A selective representation of some system that captures accurately and concisely all of its essential properties of interest for a given set of concerns*



- We don't see everything at once
- What we do see is adjusted to human understanding

Why Do Engineers Build Models?

- **To understand**
  - …the interesting characteristics of an existing or desired (complex) system and its environment
- **To predict**
  - …the interesting characteristics of the system by analysing its model(s)
- **To communicate**
  - …their understanding and design intent (to others and to oneself!)
- **To specify**
  - ...the implementation of the system (models as blueprints)

The Trouble with Models (Abstraction)

- "The devil is in the details"
- Leaving out something crucial
  - Underestimating its relevance
  - Accidental oversight

⇒ Inaccurate and untrustworthy models

WARNING: Don't confuse abstraction with reality: It doesn't always work

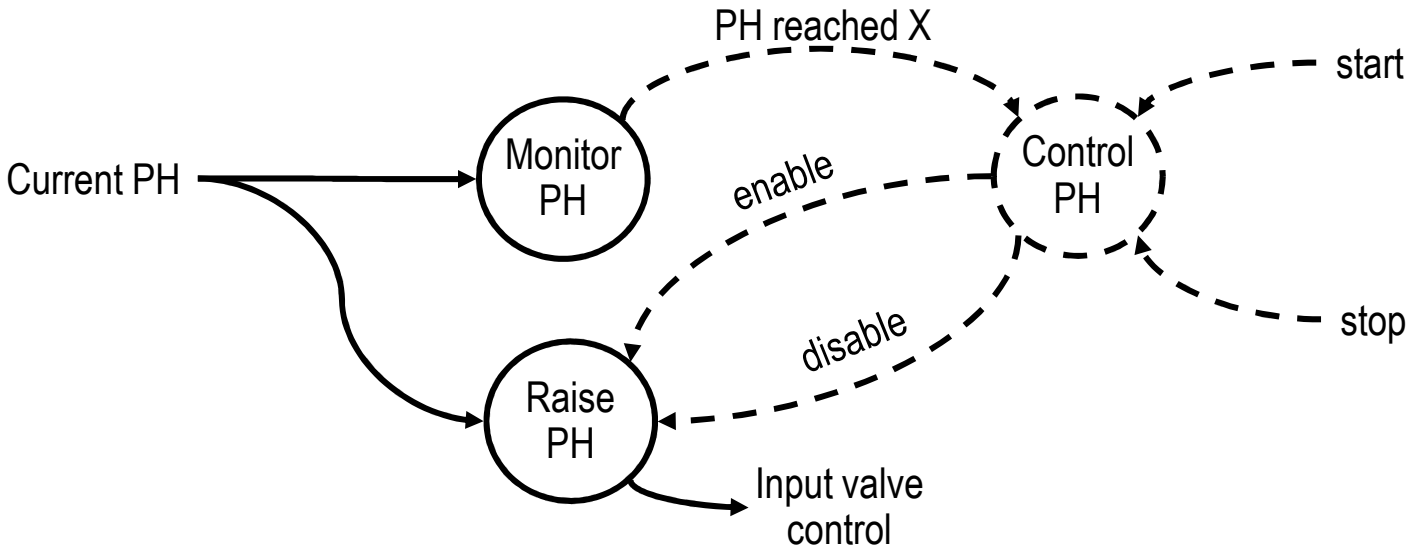
Exacerbated by the common practice of NOT documenting the abstraction process

Characteristics of Useful Engineering Models

- **Purposeful:**
  - Constructed to address a specific set of concerns/audience
- **Abstract**
  - Emphasize important aspects while removing irrelevant ones
- **Understandable**
  - Expressed in a form that is readily understood by observers
- **Accurate**
  - Faithfully represents the modeled system
- **Predictive**
  - Can be used to answer questions about the modeled system
- **Cost effective**
  - Should be much cheaper and faster to construct than actual system

To be useful, engineering models must satisfy at least these characteristics!

What About Software Modeling?

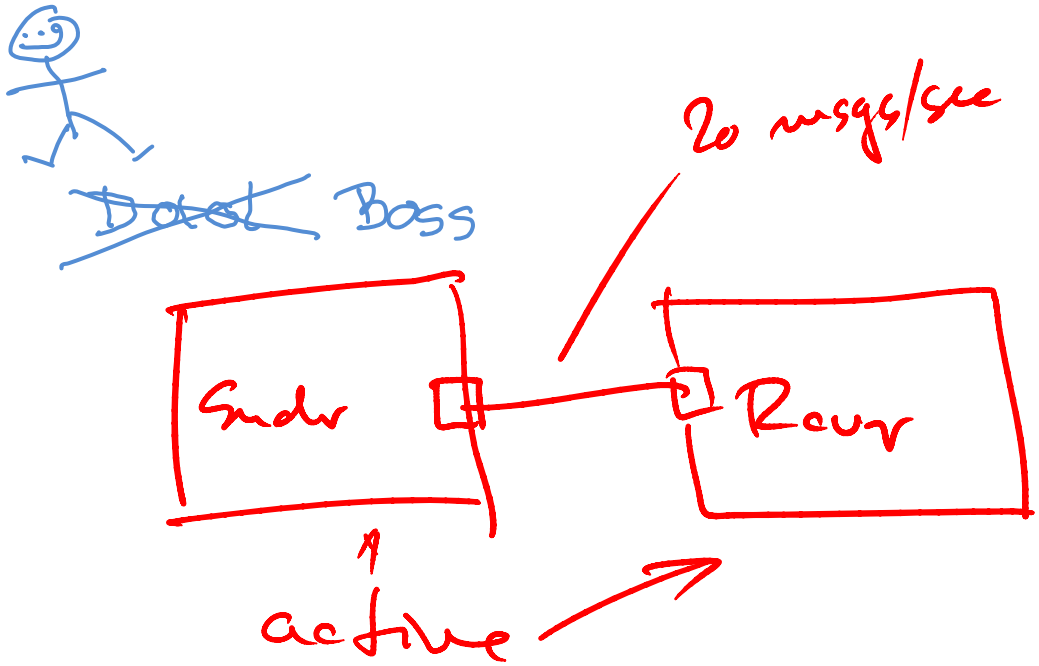


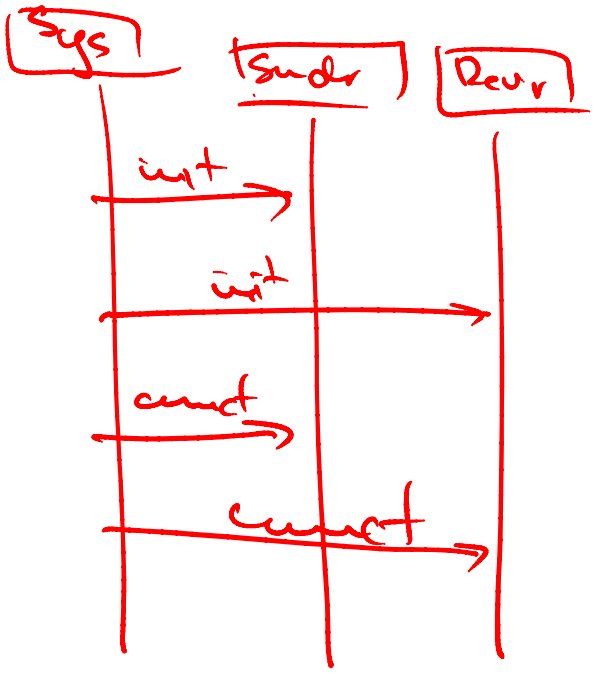
"...bubbles and arrows, as opposed to programs, ...never crash"

-- B. Meyer
"UML: The Positive Spin"
American Programmer, 1997

The Prevalent Attitude to Modeling

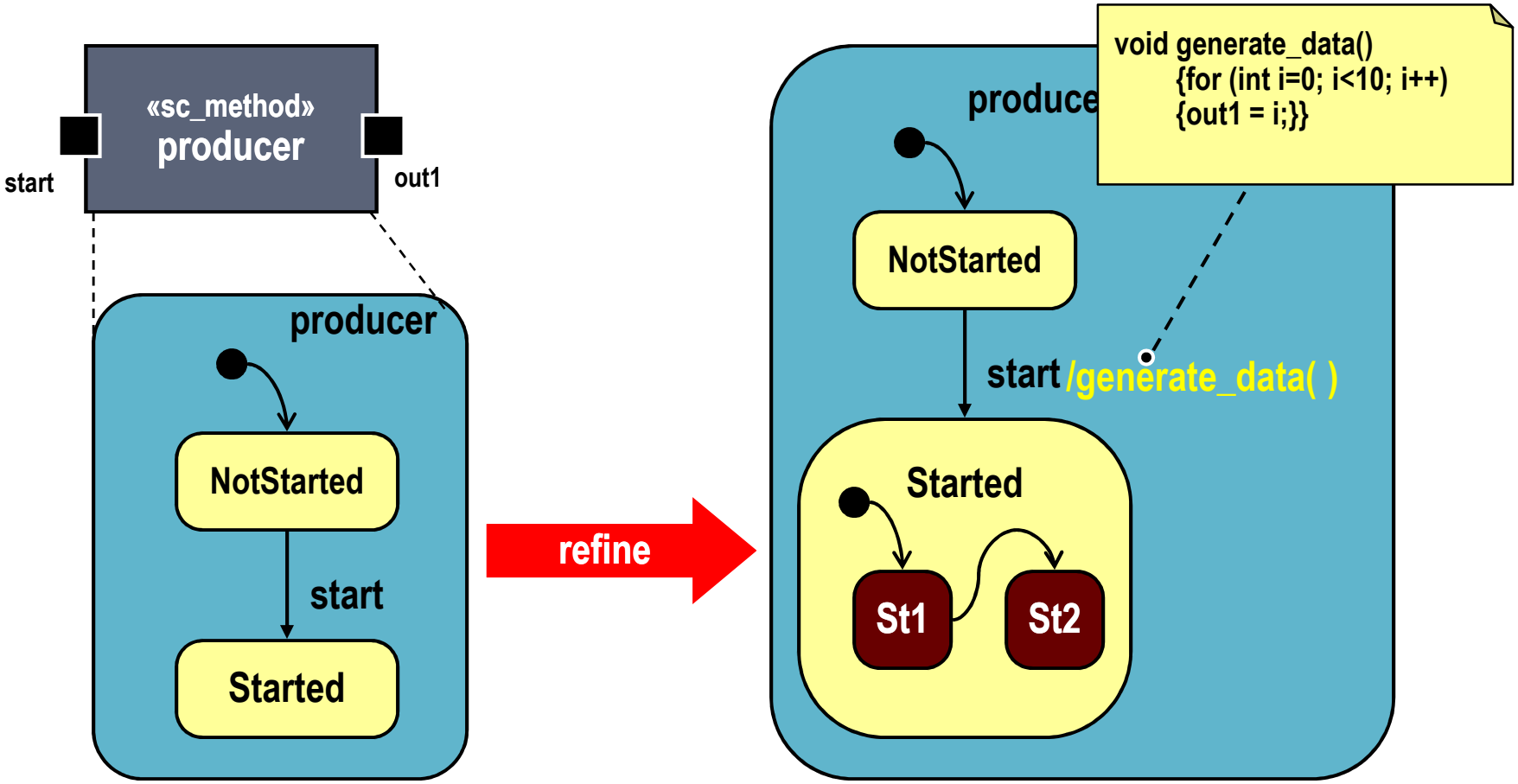
- We don't do modeling here...it's a waste of time
- But...





Modern Model-Based Software Engineering

- Models can be refined continuously until the application is fully specified ⇒ ***the model becomes the system that it was modeling!***



A Unique Feature of Software

- **A software model and the software being modeled share the same medium—the computer**
  - Which also happens to be our most advanced and most versatile automation technology

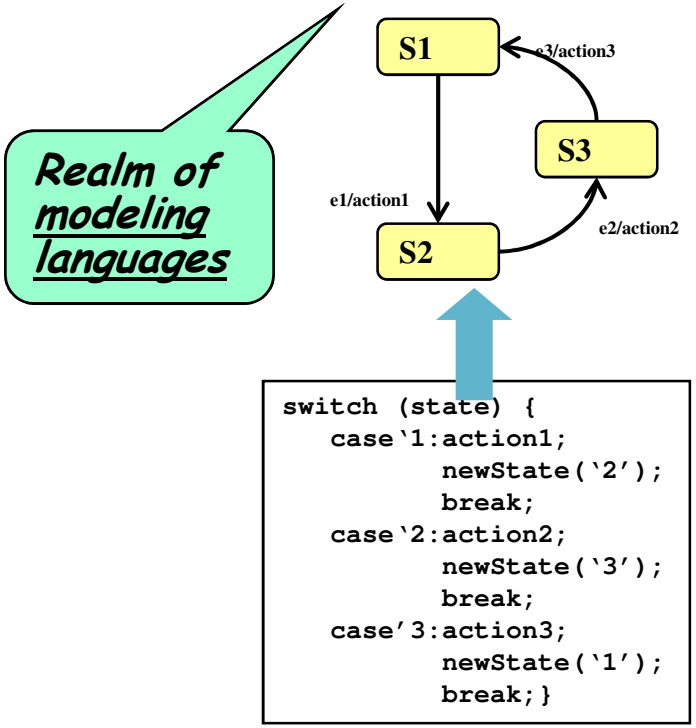
Software has the unique property that it allows us to directly evolve models into implementations without fundamental discontinuities in the expertise, tools, or methods!

⇒ *High probability that key design decisions will be preserved in the implementation and that the results of prior analyses will be valid*

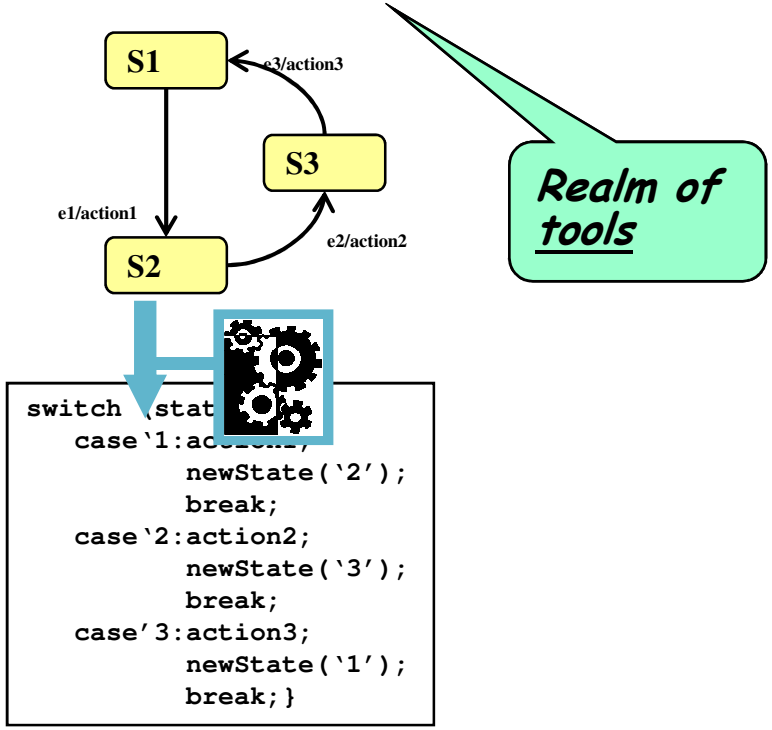
The Model-Based Engineering (MBE) Approach

- An approach to system and software development in which software models play an indispensable role
- Based on two time-proven ideas:

(1) ABSTRACTION

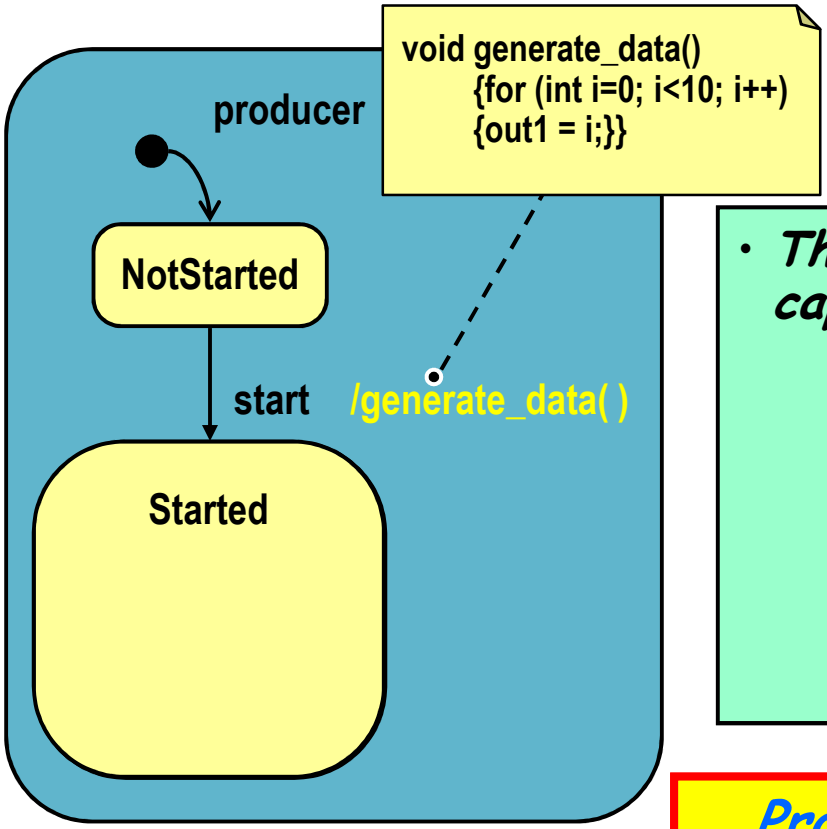


(2) AUTOMATION



But, if the Model is the System…

- …do we not lose the abstraction value of models?



- *The computer offers a uniquely capable abstraction device:*

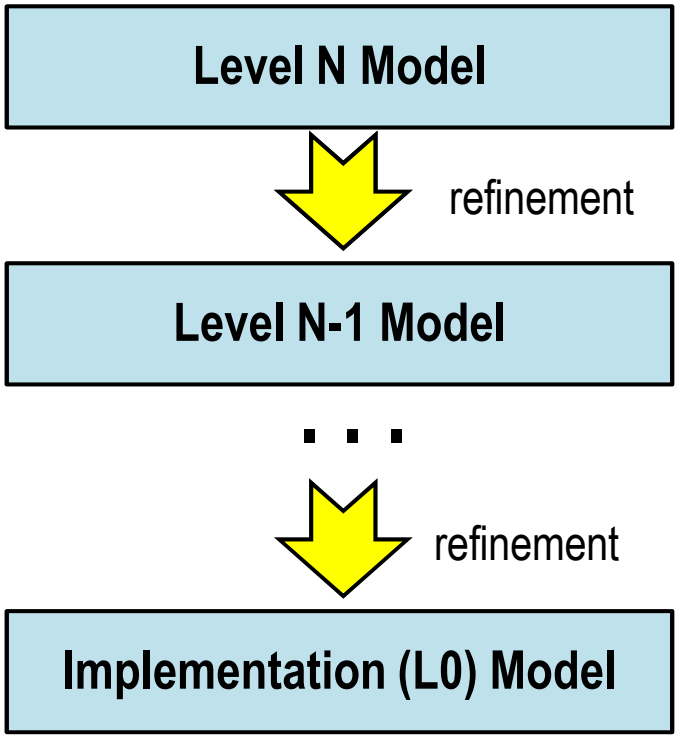
  Software can be represented from any desired viewpoint at any desired level of abstraction

  The abstraction is inside the system and can be extracted automatically via suitable model transformations

Provided that the abstraction process is tracked and recorded

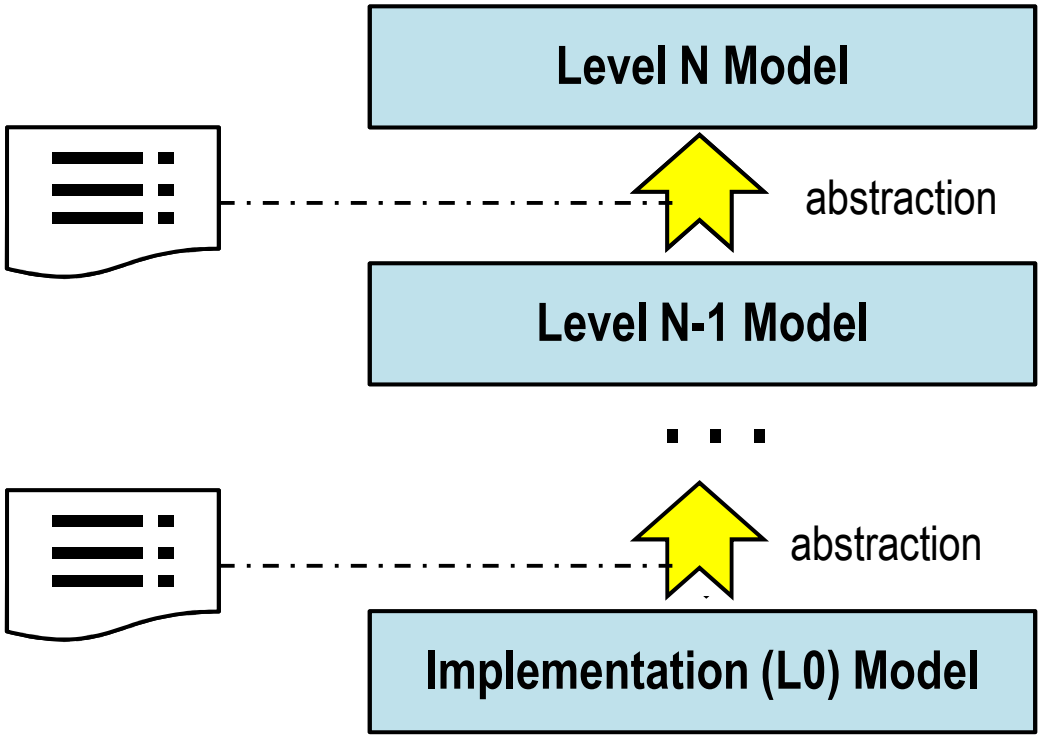
Successive Levels of Abstraction

- In practice, higher-level models are needed at all times both during and following development
  - Not just during design but also for maintenance and system evolution



Tracking Refinement/Abstraction

- If the relationships between models at different levels of abstraction are explicitly tracked and recorded, it is possible to:
  - Validate the abstraction/refinement steps
  - Reconstruct (automatically) an abstract model from a more concrete one



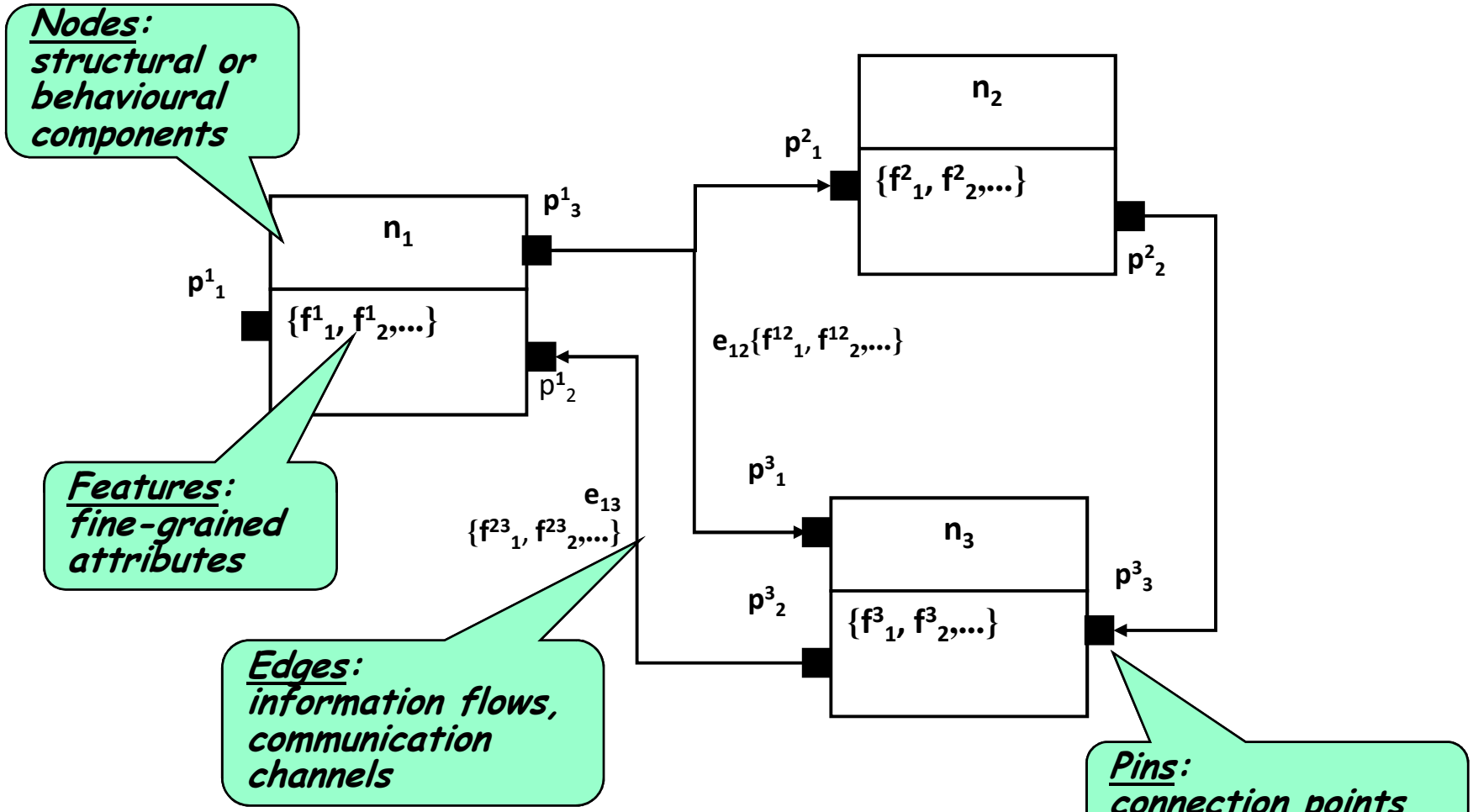
Overview

- Part I: On abstraction and modeling
- Part II: A starter catalogue of abstraction patterns for system architects

The Abstraction Patterns Catalogue

- Intended for architects and developers
- Based on a precise definition of patterns expressed using a graph-based formalism (featured graphs)
  - Suitable for various graph-based modeling languages such as UML
- Patterns for:
  - Structure
  - Behaviour
  - Time

A Provisional Formalism: Featured Graphs



- FG = <Nodes, Edges>
- n ∈ Nodes; n = <id, Ftrs(n), Pins(n)>
- e ∈ Edges; e = <id, Ftrs(e), Srcs(e), Dests(e)>

Abstraction/Refinement Patterns

- Each pattern consists of a refinement graph, an abstract graph, and a formally defined set of mappings between them

  Pattern = <RefGraph, Mappings, AbsGraph>

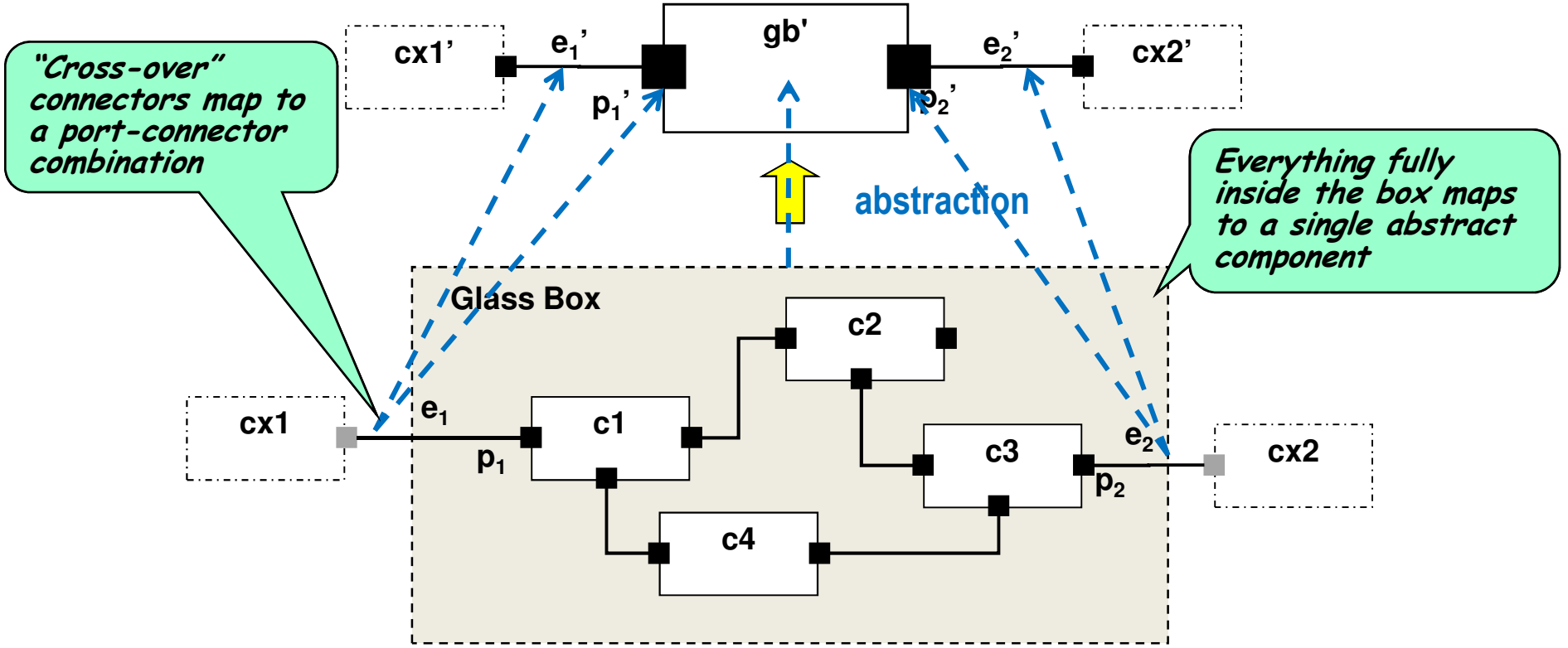
- Each mapping is a pair comprising a single AbsGraph element and a corresponding RefGraph element

  $\forall$refElem $\in$ RefGraph | $\exists$mapping $\in$ Mappings
  mapping = <absElem, refElem>
  absElem $\in$ AbsGraph

- *Each refGraph element is covered by a mapping to ensure that nothing is overlooked*

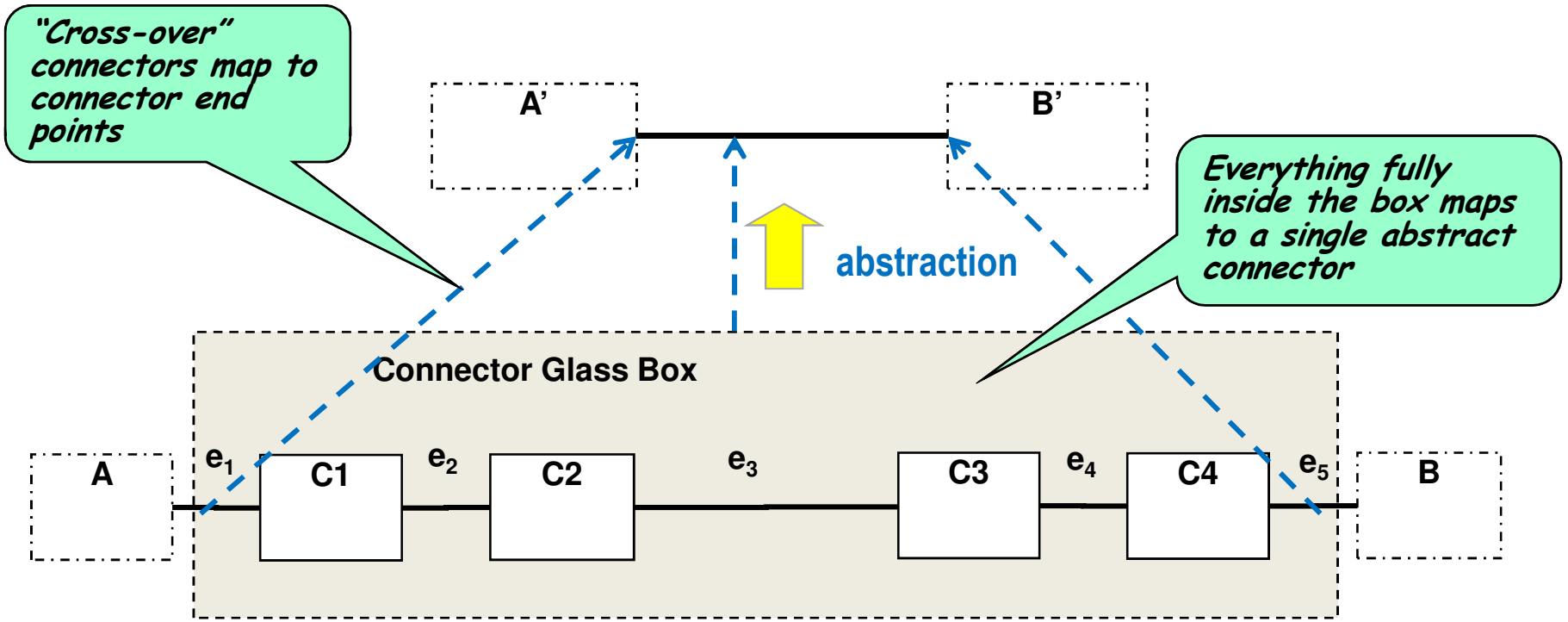
Structural Pattern: Black Box

- Based on a common meta-pattern that appears in different forms in both structure and behaviour modeling
  - Synthesizes a network of tightly-coupled concrete components and renders them as a single high-level component



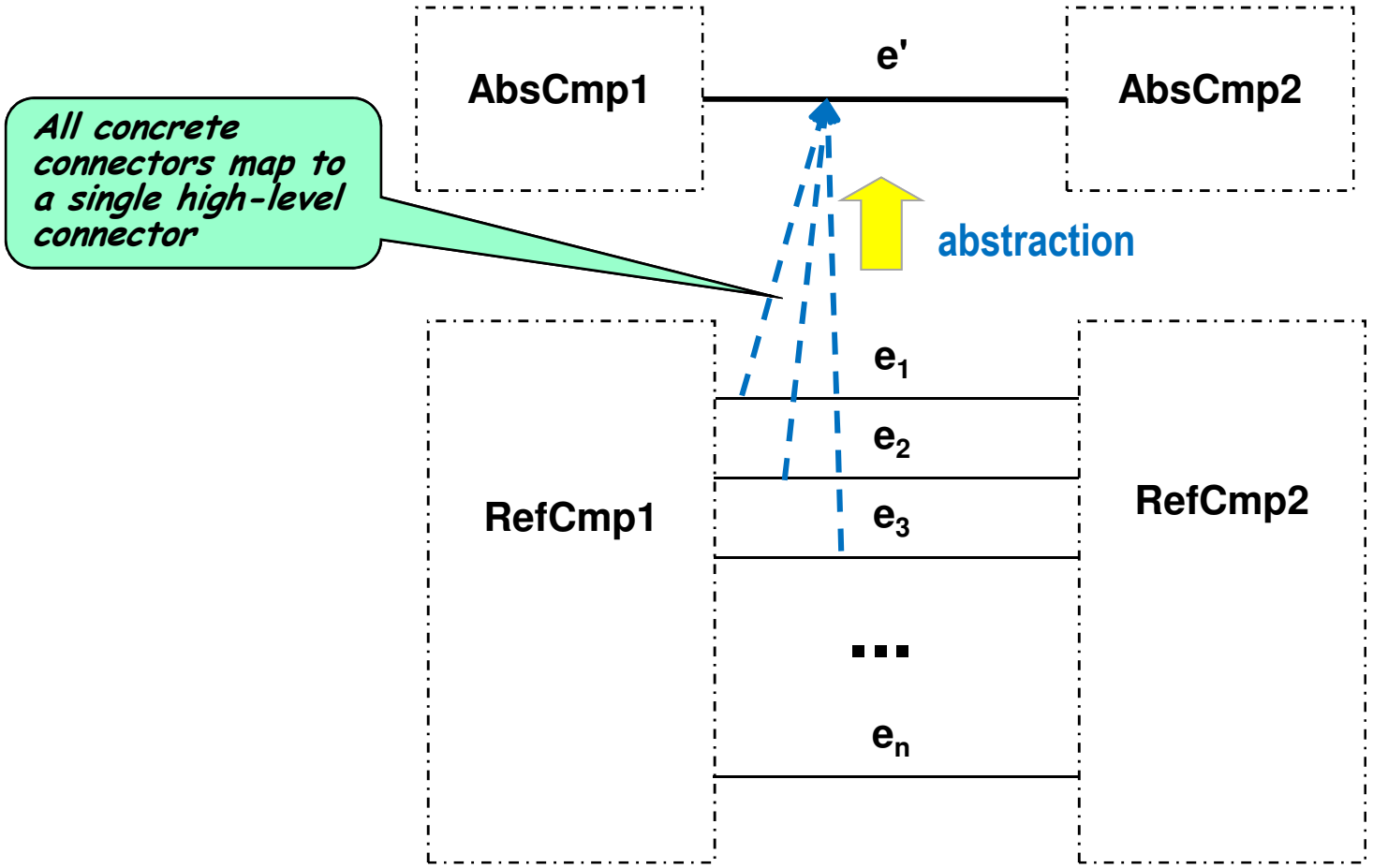
Structural Pattern: Black Line

- Abstracts a collection of elements realizing a communications path into a single edge (connector)
  - Could be multipoint



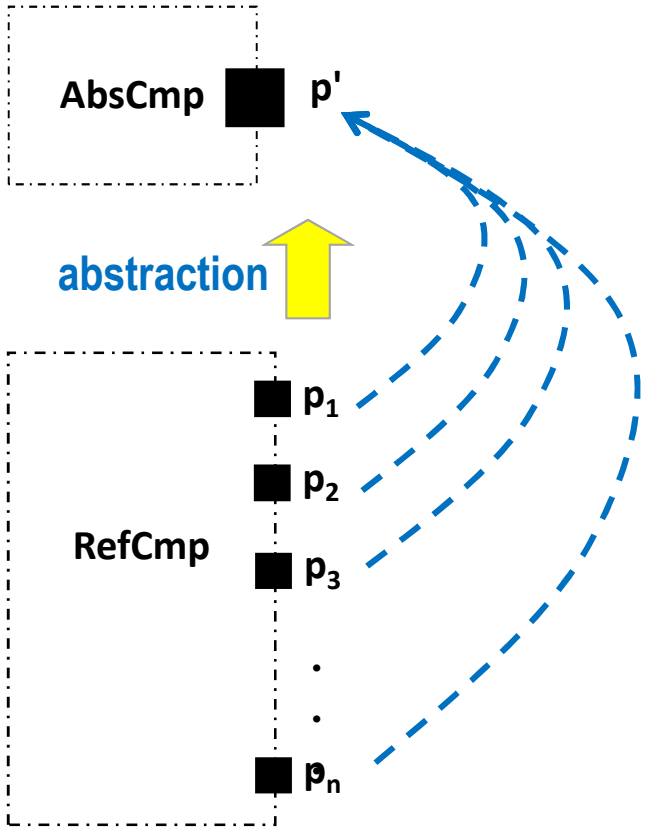
Structural Pattern: Cable

- Group of connectors that share the same source and sink components



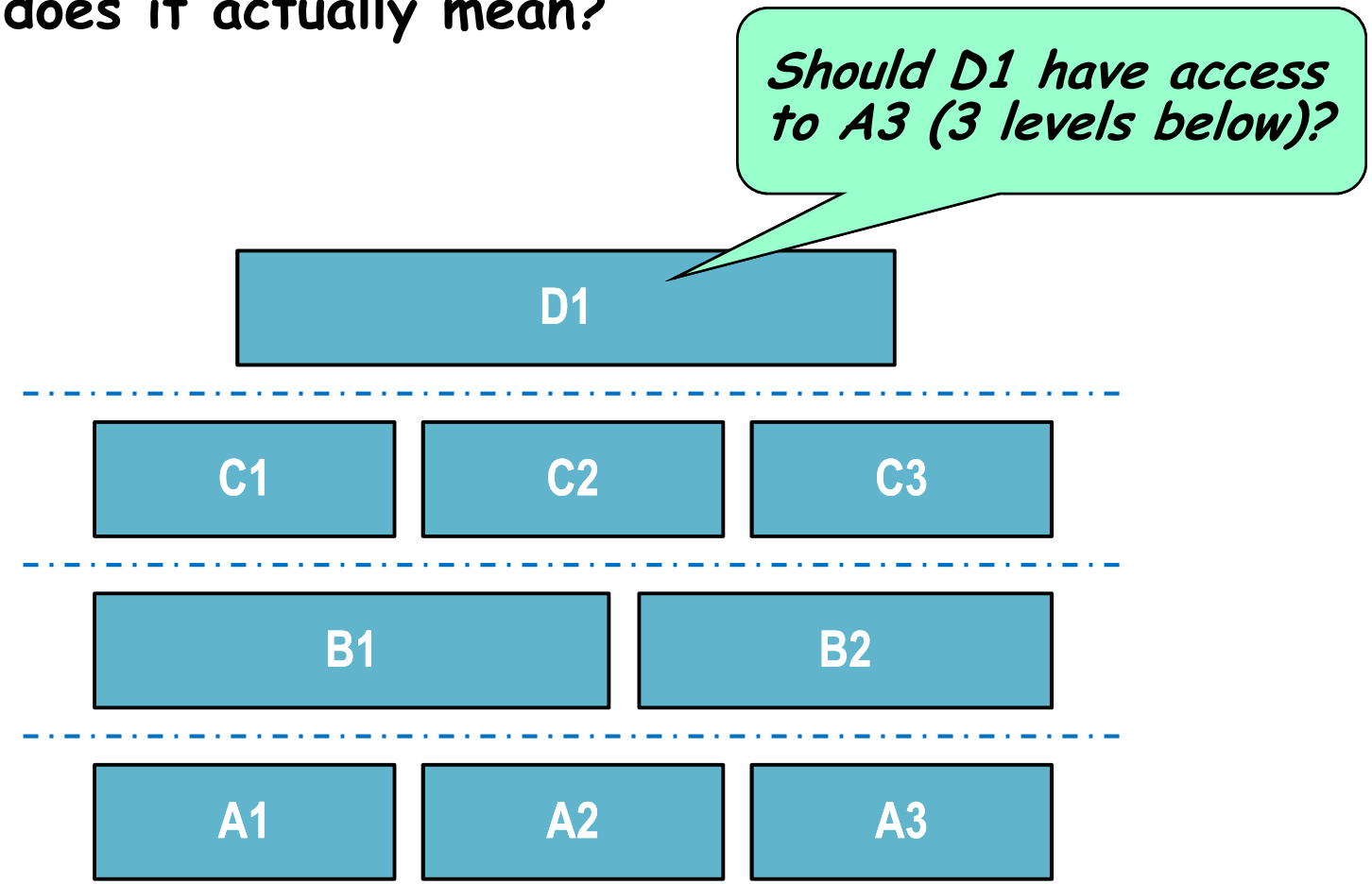
Structural Pattern: Port Group

- Multiple concrete ports (possibly different types) merged into a single abstract port
  - Often combined with "Cable" pattern

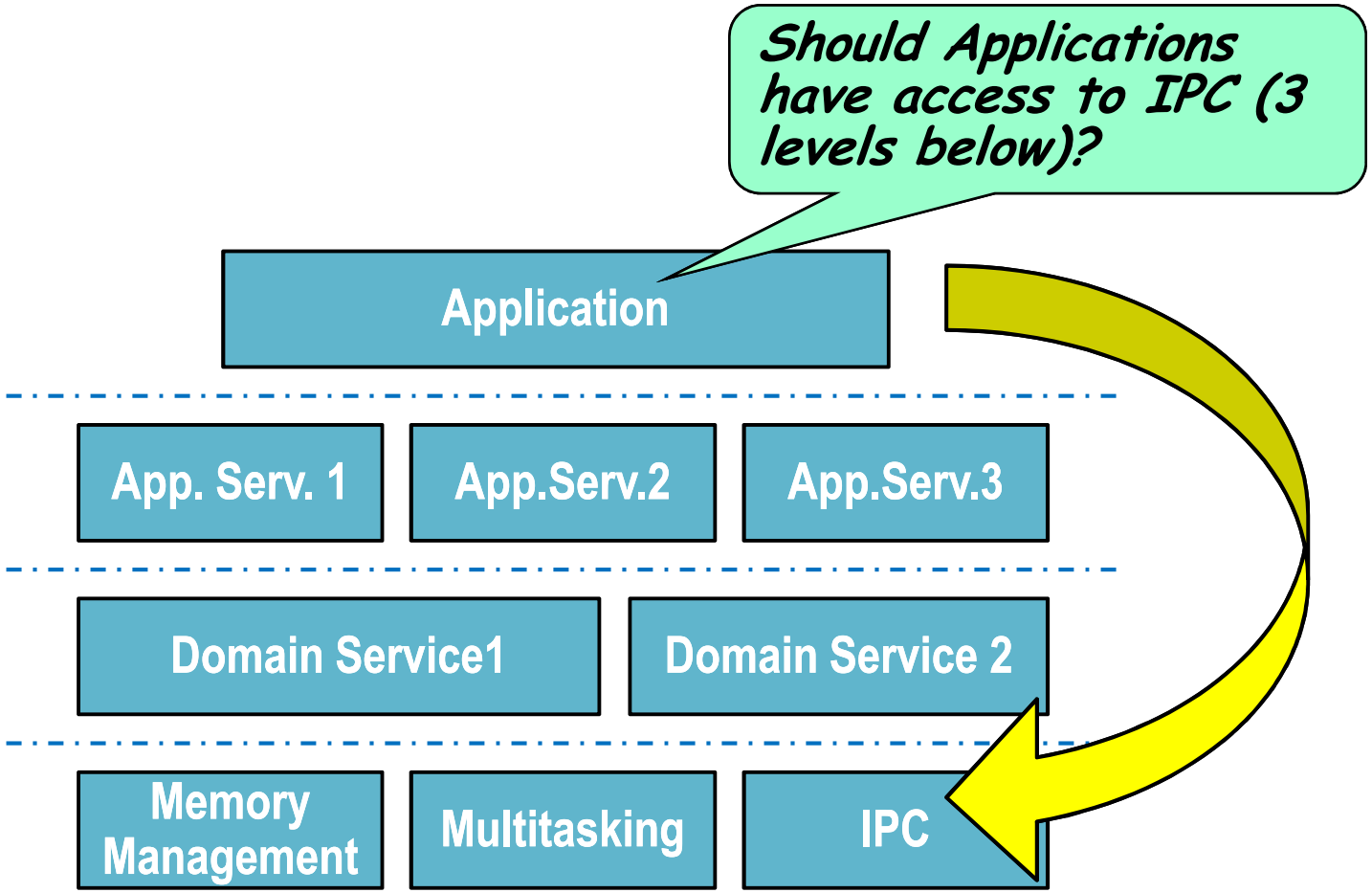


A Common Structural Pattern: Layering

- What does it actually mean?

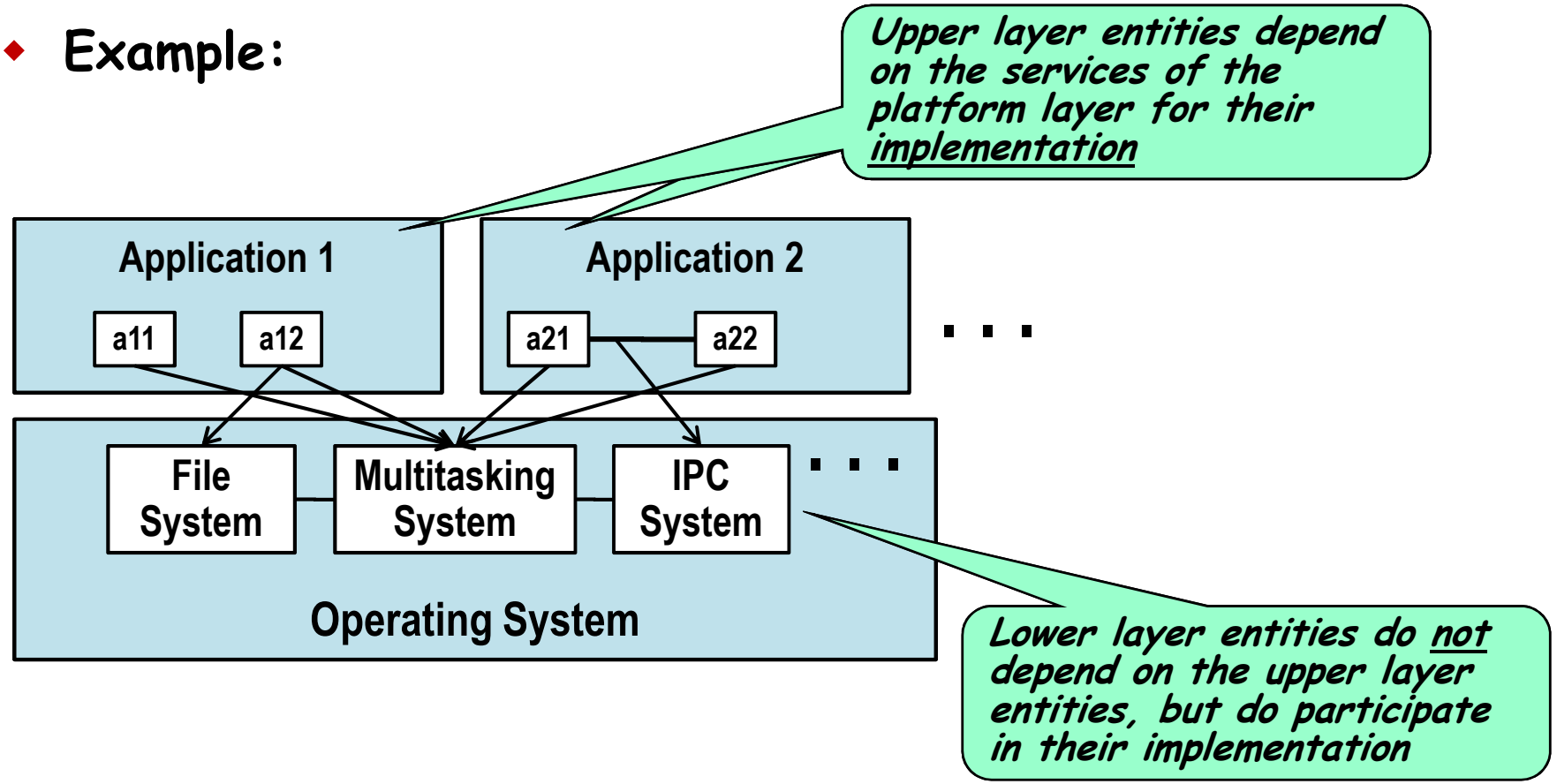


Making it a Bit More Concrete



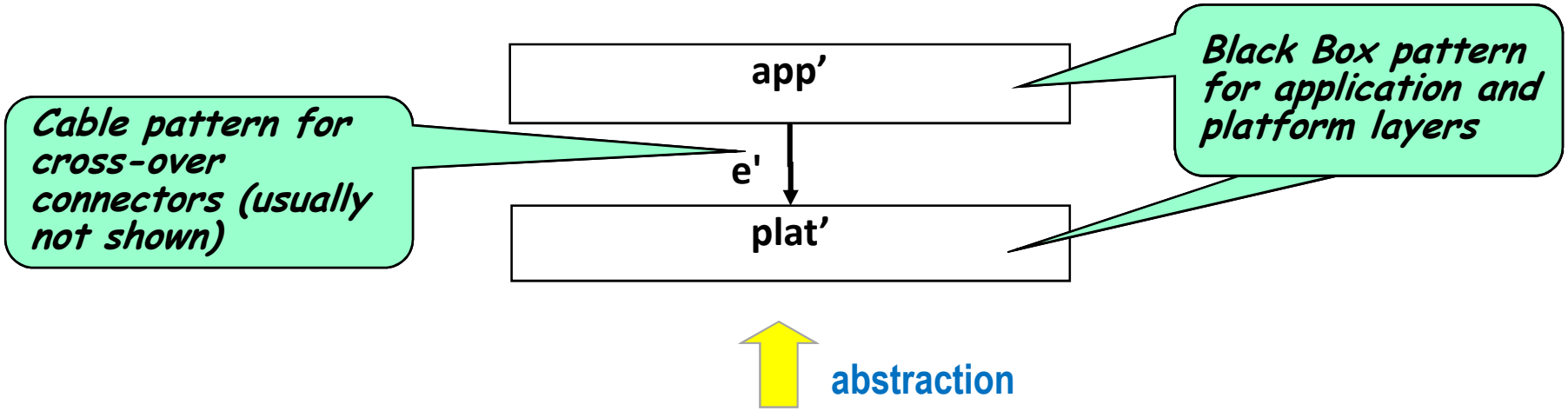
"Platform" Layering

- Distinguishing characteristics
  - Upper layer: realizes some higher-level functionality
  - Lower layer: a set of potentially shared services used to implement the upper layer = "platform"
- Example:



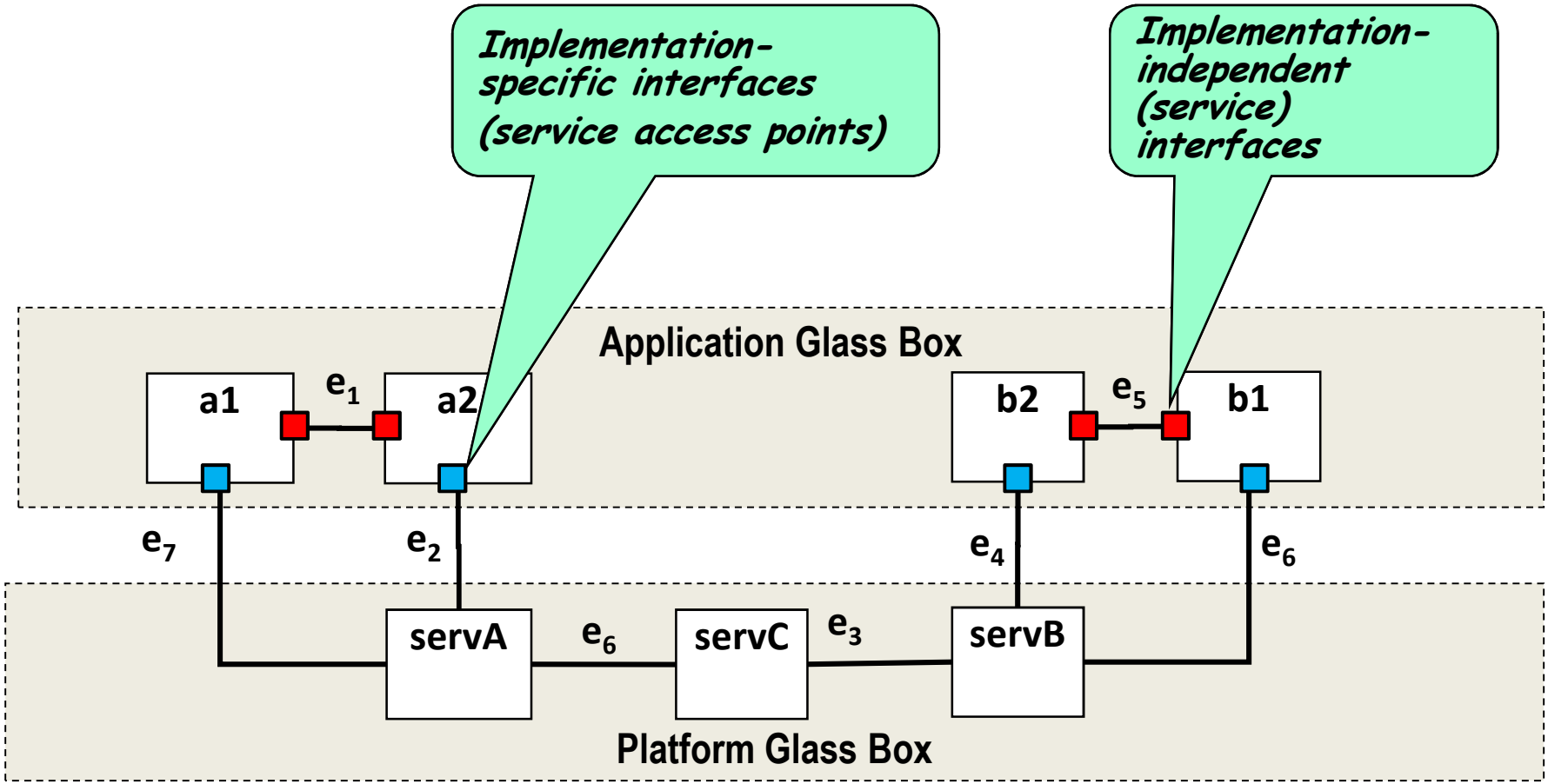
Structural Pattern: Platform Layering

- A composition of multiple pattern applications

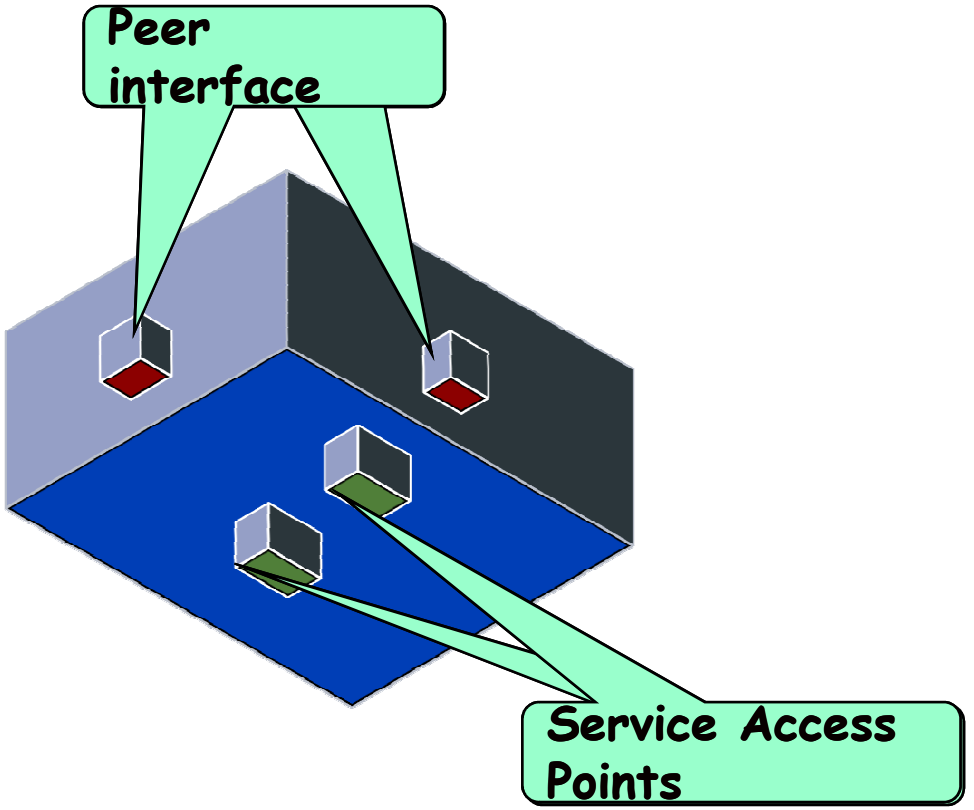


The Semantics of Layering

- Layering requires distinguishing between two categories of interfaces



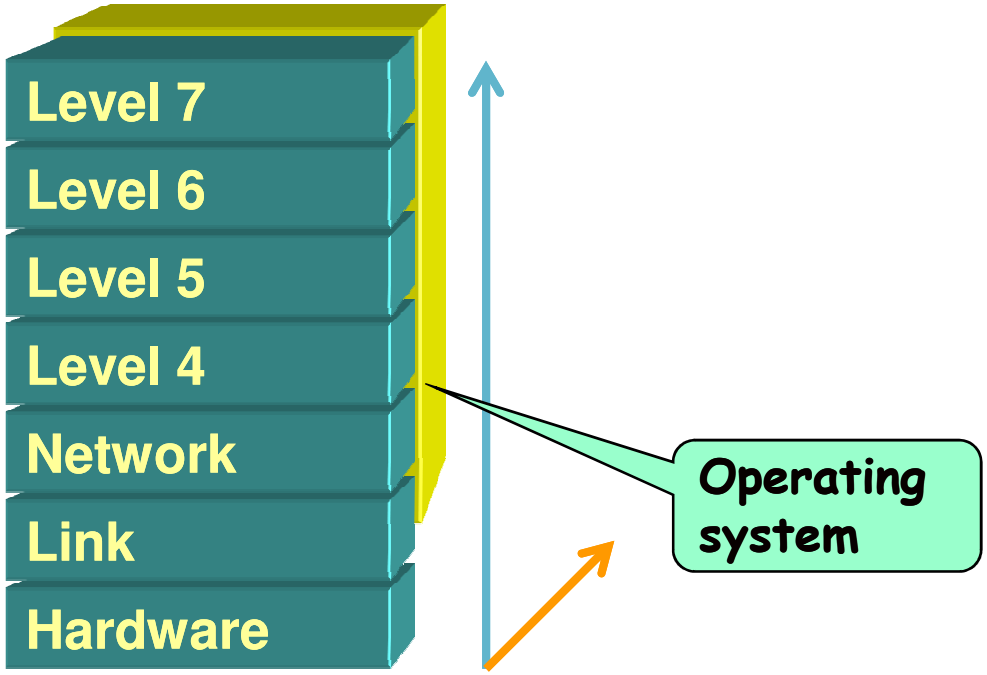
The 3-Dimensional Component Model



- *"To understand the capabilities of a black-box component it is sufficient to know its interface"*
  - *This may mean that we have to know its layer interfaces as well*

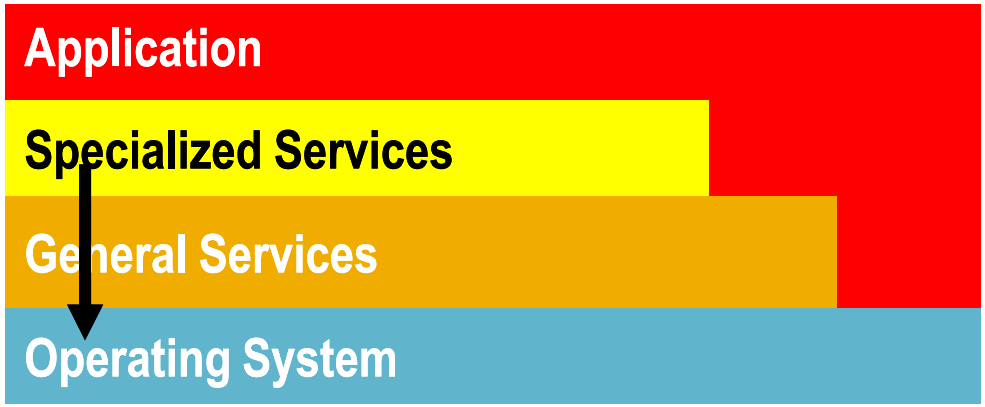
Representing Layered System Architectures

- **In most systems, layering is a complex multidimensional relationship**
  - Most real system architectures cannot be described accurately by a single vertical layer stack
  - Most vertical stacks are merely architectural views along a particular viewpoint

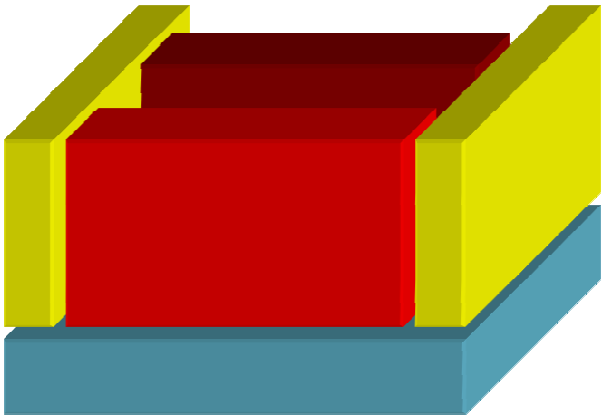


Failed Representations of Layering

- Staircase model

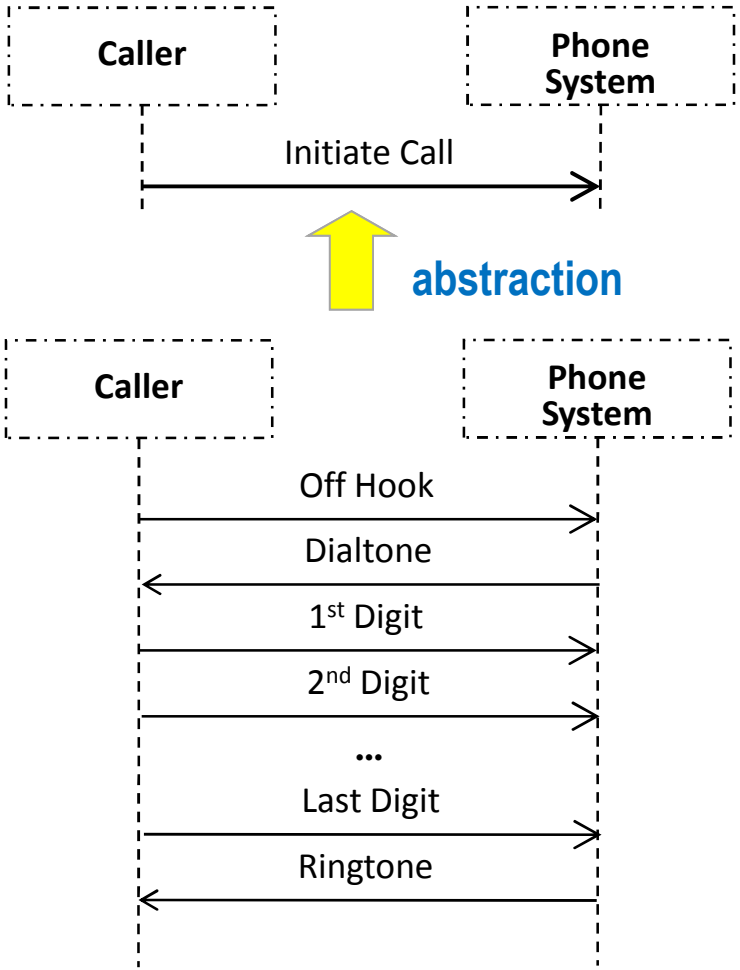


- Toaster model



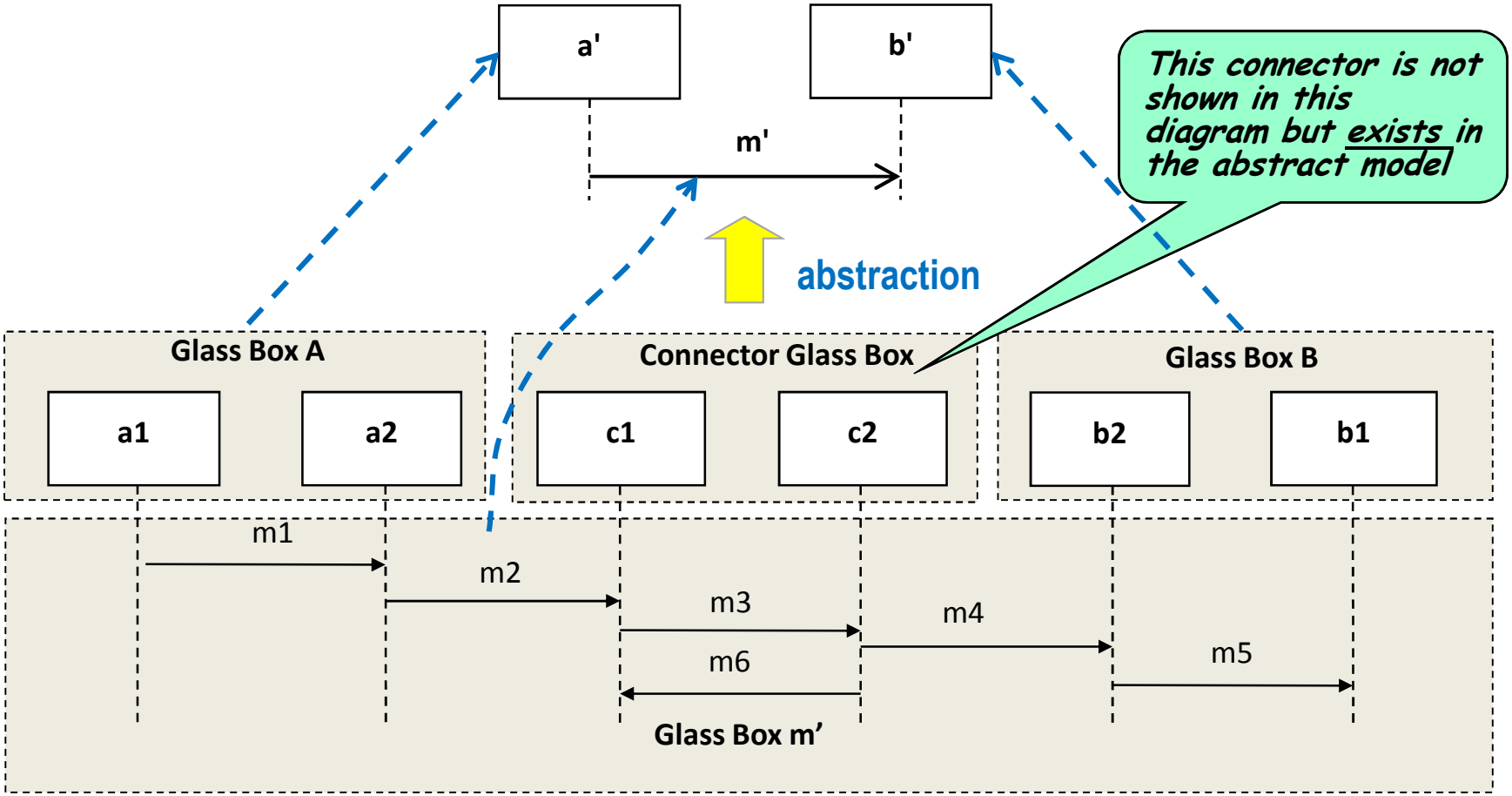
Behavioural Pattern: Summary Message (1)

- A high-level message abstracts a low-level protocol
  - Syntactically equivalent to the Cable pattern



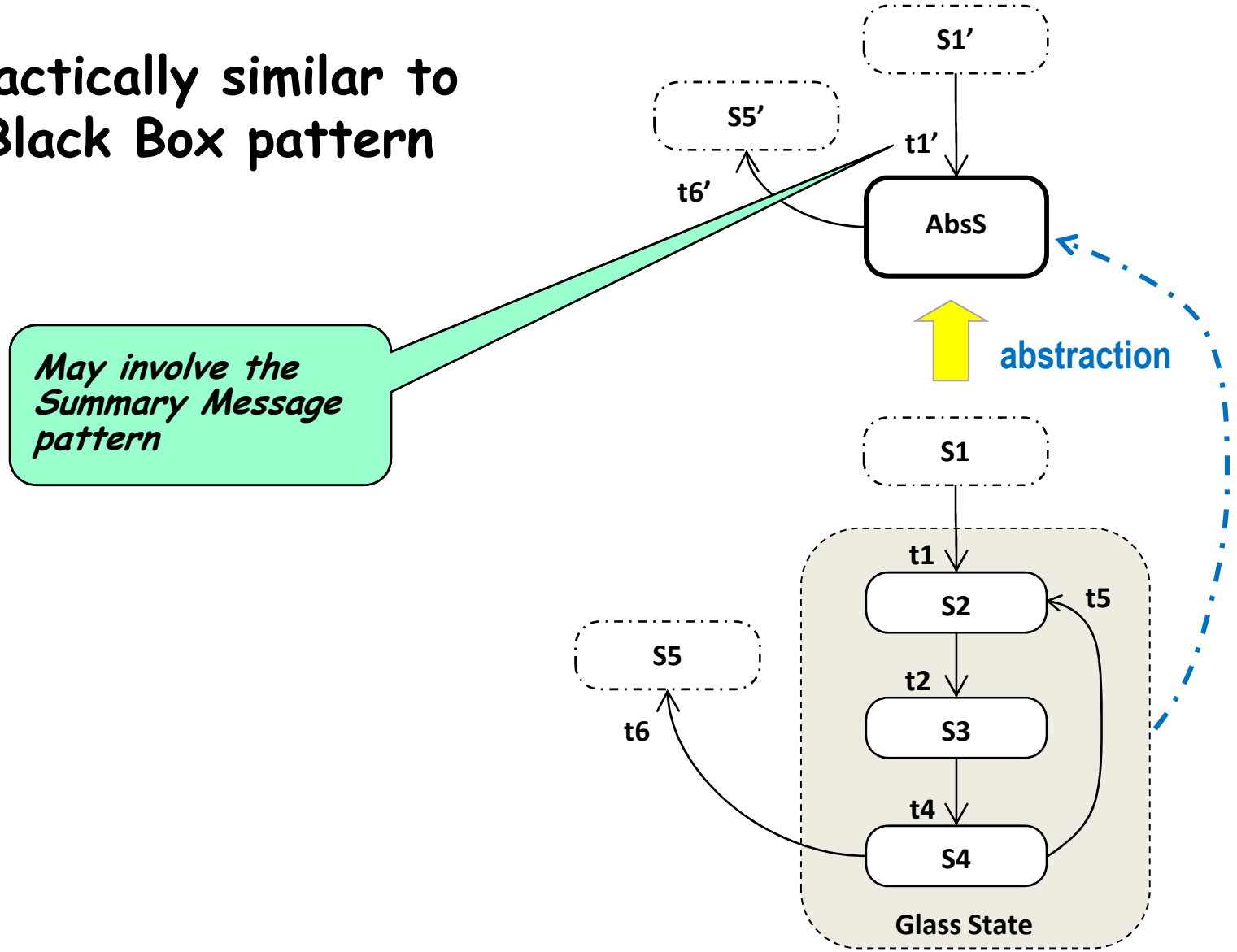
Hybrid Pattern: Layered Communication

- Combination of structural and behavioural patterns



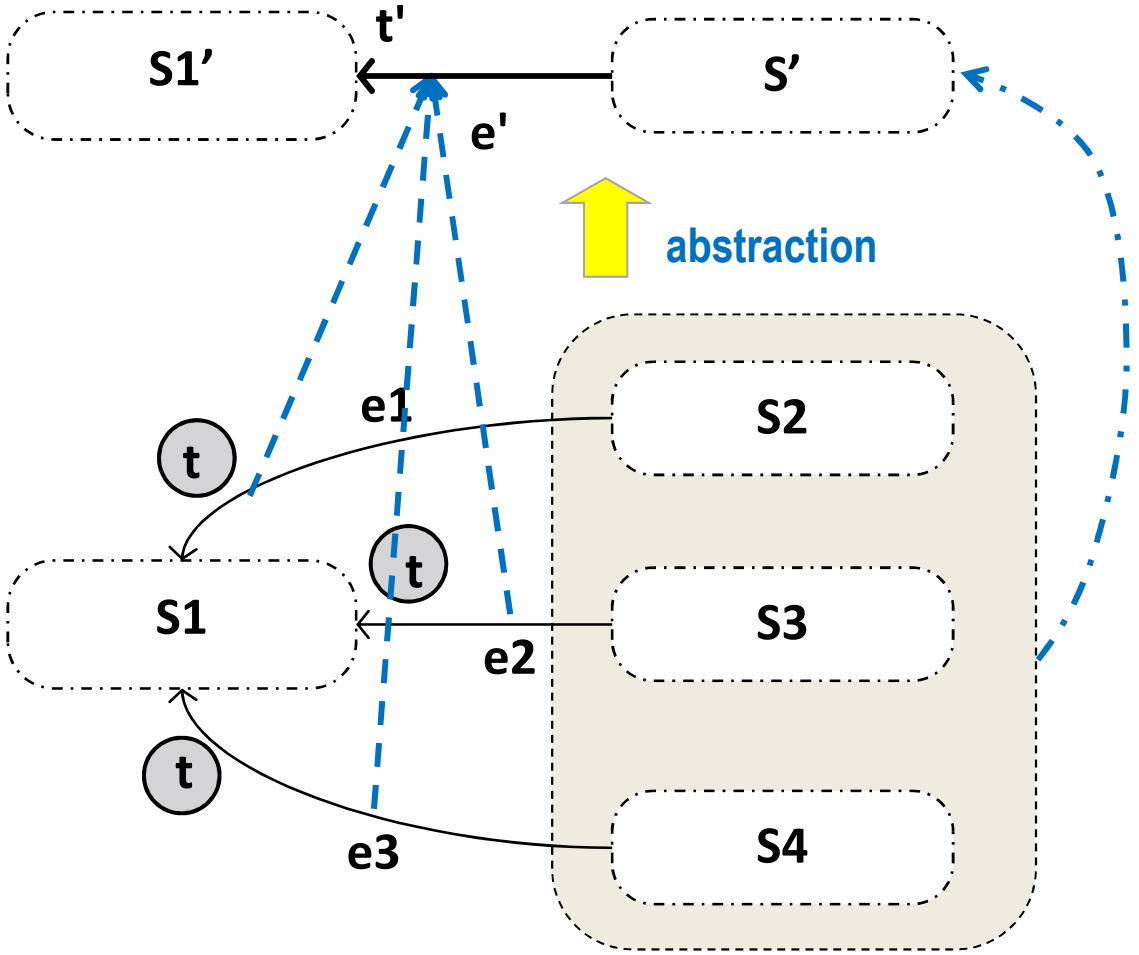
Behavioural Pattern: Summary State

- Syntactically similar to the Black Box pattern



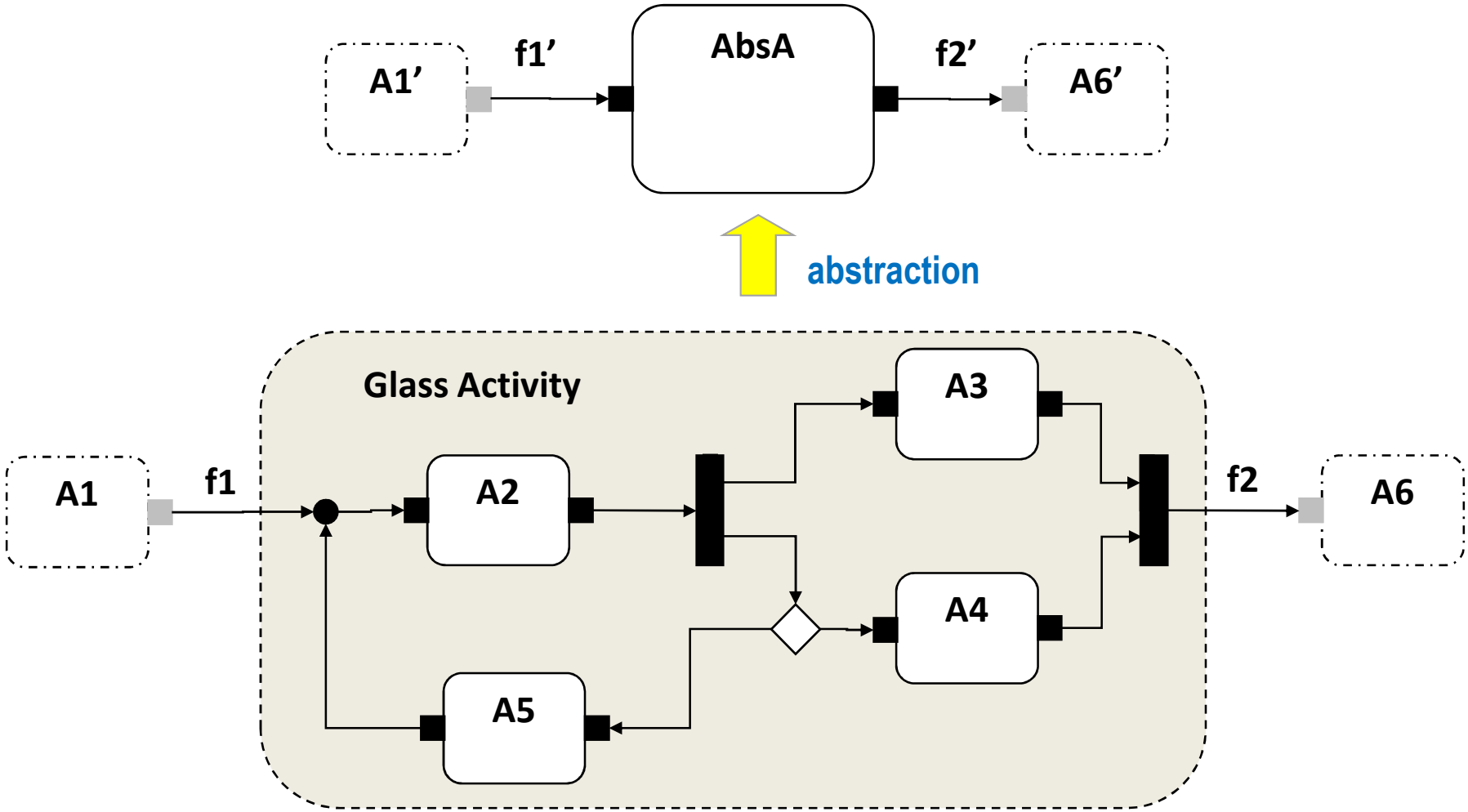
Behavioural Pattern: Group Transition

- Involves a Summary State pattern followed by a syntactical equivalent of the structural Cable pattern



Behavioural Pattern: Summary Activity

- Syntactical equivalent of Black Box



Temporal Abstraction Patterns

- **For reducing the complexity of time**
  - Not inherently graph based
- **Time Compression**
  - Duration abstracted as an instant
- **Logical Time**
  - Duration abstracted away entirely
  - Only temporal order is preserved
- **Discrete Time**
  - Time discretized into equidistant intervals
  - Often combined with time compression: all behaviour is compressed to the instant when the current interval ends

Summary

- Abstraction is the primary tool for dealing with overwhelming complexity
- With model-based engineering methods, we are taking greater advantage of the power of abstraction in software development…but, the actual process involved is rarely documented

⇒ Difficult to identify faulty abstractions and faulty refinements

⇒ …and difficult to reconstitute abstract views of an implemented system

- A formalism for describing the abstraction process combined with a catalogue of widely used and proven abstraction patterns can help us mitigate and possibly eliminate these issues

References

- B. Selic, "A Short Catalogue of Abstraction Patterns for Model-Based Software Engineering", *Int. Journal of Software Informatics,* vol. 5, issue 2, 2011 (in publication).
- Dirgahayu T, Sinderen MV,Quartel D. Abstractions of interaction mechanisms. 2009 IEEE International Enterprise Distributed Object Computing Conference (EDOC 2009). IEEE, 2009 (173-182).
- Giunchiglia F, Villafiorita A, Walsh T. Theories of Abstraction. Technical Report MRG/ DIST# 97 0051. Universita di Genova, Facolta di Ingengneria, 1997.
- Knoblock CA. Automatically generating abstractions for planning. Arti¯cial Intelligence, 1994, 68: (243-302).
- Long RT. Realism and abstraction in economics: Aristotle and Mises versus Friedman. Quarterly Journal of Austrian Economics. SpringerLink, 2008, 9(3): (3-23).
- Atkinson C, KÄuhne T. Aspect-oriented development with strati¯ed frameworks. IEEE Software, Jan.-Feb. 2003: (81-89).
- Boiten E, Derrick J. Proc. of the 14th BCS-FACS Re¯nement Workshop (REFINE 2009). Electronic Notes in Computer Science, Eindhoven, Elsevier, 2009.

- THE EGRESS -

QUESTIONS, COMMENTS, ARGUMENTS...